Certification Program in
Business Analytics & Optimisation
From IIT DELHI

TERM PROJECT

TOPIC: CREDIT CARD FRAUD DETECTION

From:

Shivesh Kumar Sareen
(shivesh72@gmail.com)

INTRODUCTION

Fraud detection is a set of activities undertaken to detect malicious activities and prevent money and property from being taken through false means. One of the common frauds in the banking and financial sector is the **Credit Card Fraud**. Therefore, analysis tools need to be employed to detect such occurrences and prevent them to the extent possible. This can be accomplished by identifying suspicious events and reporting them to an analyst, while allowing regular transactions to be automatically processed.

PROBLEM STATEMENT

As an analyst in a banking or financial sector, you are entrusted with the task to detect potential fraud cases. This helps in gaining customer confidence as they are ensured that they will not be charged for the items, not purchased by them. Based on the input dataset, containing details like transaction between people, machine learning model need to be built to differentiate whether the transactions are fraud or not.

The dataset contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

METHODOLOGY

To achieve this objective, different classification models need to be built to classify and distinguish fraud transactions.

Since, data is available for building the Machine learning model, it's a case of structured data analysis. Also, the output has to be in terms of "Yes" or "No", to determine whether transaction is a fraud or not. Therefore, classification approach will be used instead of Regression.

Classification is the process of predicting discrete variables (0/1, Yes/No, etc.).

STEPS INVOLVED

1. Import all the required libraries like Pandas, Numpy, Sklearn etc.
2. Import the dataset to be analysed.
3. Define the input and output variables.

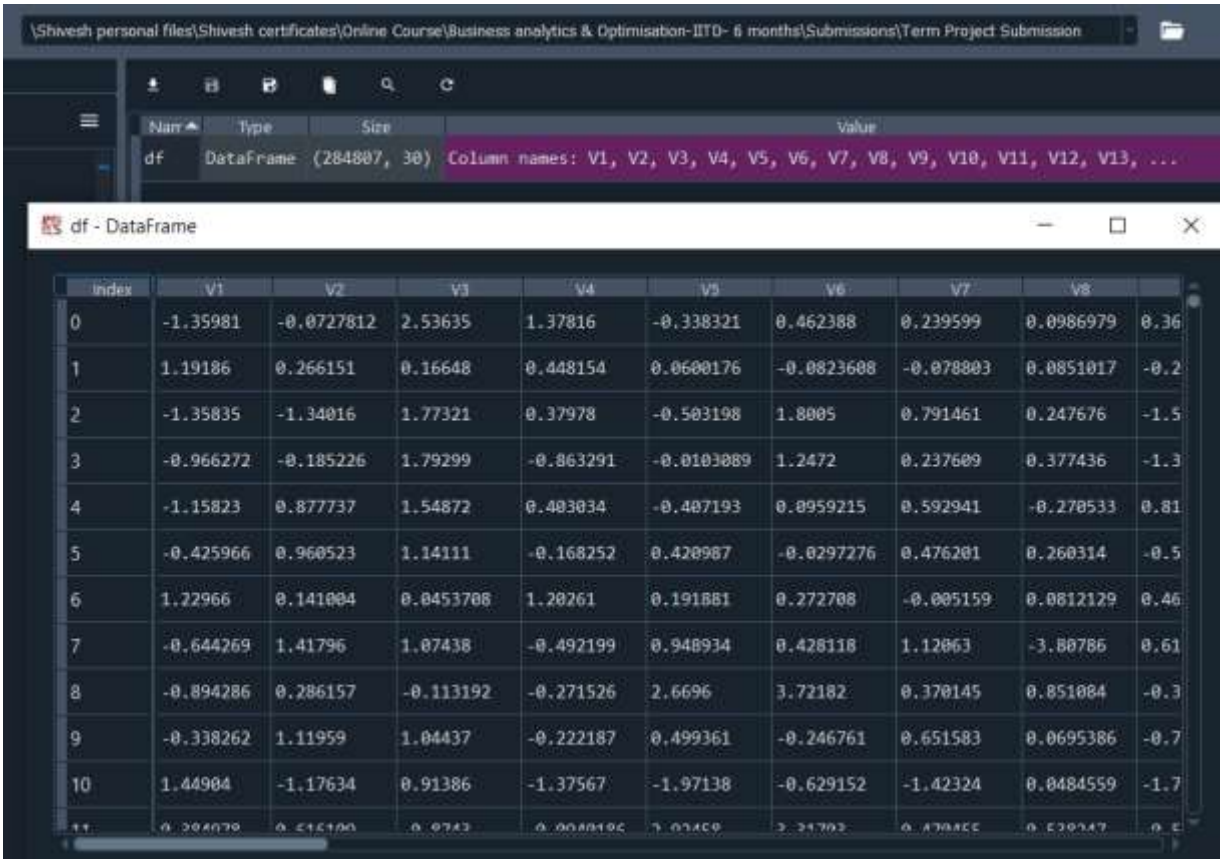
4. Pre-processing the data and exploratory data analysis.
5. Splitting the data into training and test data
6. Perform feature scaling, if required
7. Develop the classification models.
8. Evaluate the developed classification models.

Step 1: Import all the required libraries

```
# IMPORTING PACKAGES

# IMPORTING BASIC PACKAGES

import pandas as pd                          # data processing
import numpy as np                           # working with arrays
import matplotlib.pyplot as plt              # visualization

# IMPORTING PACKAGES for PREPROCESSING

from sklearn.preprocessing import StandardScaler          # data normalization
from sklearn.model_selection import train_test_split      # data split

# IMPORTING PACKAGES for building CLASSIFICATION MODEL

from sklearn.tree import DecisionTreeClassifier           # Decision tree algorithm
from sklearn.neighbors import KNeighborsClassifier        # KNN algorithm
from sklearn.linear_model import LogisticRegression       # Logistic regression algorithm
from sklearn.svm import SVC                               # SVM algorithm
from sklearn.ensemble import RandomForestClassifier # Random forest tree algorithm

# IMPORTING PACKAGES for evaluating CLASSIFICATION MODEL

from sklearn.metrics import confusion_matrix              # evaluation metric
from sklearn.metrics import accuracy_score                # evaluation metric
from sklearn.metrics import f1_score                      # evaluation metric
```

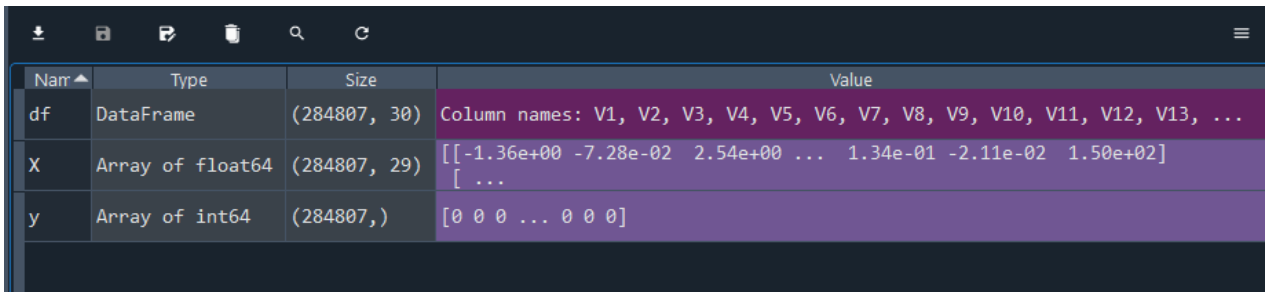
Step 2: Import dataset to be analysed

```
df = pd.read_csv('creditcard.csv')
df.drop('Time', axis = 1, inplace = True)
print(df.head())
```



index	V1	V2	V3	V4	V5	V6	V7	V8	
0	-1.35981	-0.0727812	2.53635	1.37816	-0.338321	0.462388	0.239599	0.0986979	0.36
1	1.19186	0.266151	0.16648	0.448154	0.0600176	-0.0823608	-0.078803	0.0851017	-0.2
2	-1.35835	-1.34016	1.77321	0.37978	-0.503198	1.8005	0.791461	0.247676	-1.5
3	-0.966272	-0.185226	1.79299	-0.863291	-0.0103089	1.2472	0.237609	0.377436	-1.3
4	-1.15823	0.877737	1.54872	0.403034	-0.407193	0.0959215	0.592941	-0.270533	0.81
5	-0.425966	0.960523	1.14111	-0.168252	0.420987	-0.0297276	0.476201	0.260314	-0.5
6	1.22966	0.141004	0.0453708	1.20261	0.191881	0.272708	-0.005159	0.0812129	0.46
7	-0.644269	1.41796	1.07438	-0.492199	0.948934	0.428118	1.12063	-3.80786	0.61
8	-0.894286	0.286157	-0.113192	-0.271526	2.6696	3.72182	0.370145	0.851084	-0.3
9	-0.338262	1.11959	1.04437	-0.222187	0.499361	-0.246761	0.651583	0.0695386	-0.7
10	1.44904	-1.17634	0.91386	-1.37567	-1.97138	-0.629152	-1.42324	0.0484559	-1.7

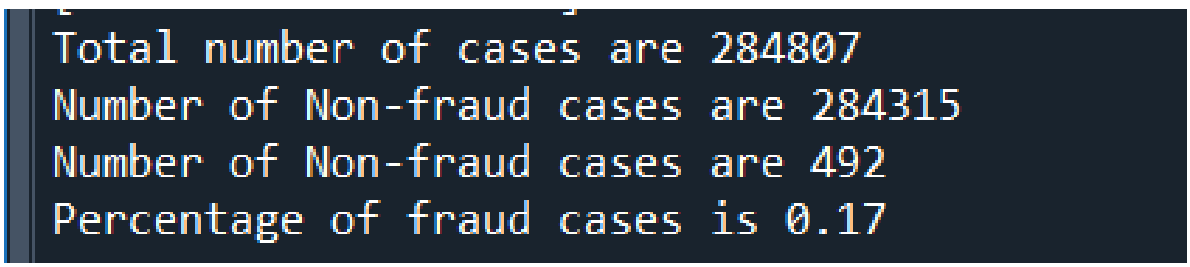
Step 3: Defining the variables (input and output)



Name	Type	Size	Value
df	DataFrame	(284807, 30)	Column names: V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, ...
X	Array of float64	(284807, 29)	[[-1.36e+00 -7.28e-02 2.54e+00 ... 1.34e-01 -2.11e-02 1.50e+02] [...
y	Array of int64	(284807,)	[0 0 0 ... 0 0 0]

Step 4: Data pre-processing and exploratory data analysis

```
print('Total number of cases are {}'.format(cases))
print('Number of Non-fraud cases are {}'.format(nonfraud_count))
print('Number of Non-fraud cases are {}'.format(fraud_count))
print('Percentage of fraud cases is {}'.format(fraud_percentage))
```



```
nonfraud_cases = df[df.Class == 0]
fraud_cases = df[df.Class == 1]
print('CASE AMOUNT STATISTICS')
print('NON-FRAUD CASE AMOUNT STATS')
print(nonfraud_cases.Amount.describe())
print('FRAUD CASE AMOUNT STATS')
print(fraud_cases.Amount.describe())
```

```
CASE AMOUNT STATISTICS
NON-FRAUD CASE AMOUNT STATS
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
FRAUD CASE AMOUNT STATS
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

Step 5: Splitting the data into training and test data

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =
0)
print('X_train samples : ', X_train[:1])
print('X_test samples : ', X_test[0:1])
print('y_train samples : ', y_train[0:10])
print('y_test samples : ', y_test[0:10])
```

```
X_train samples :  [[-1.12e+00  1.04e+00  8.01e-01 -1.06e+00  3.26e-02  8.53e-01 -6.14e-01
  -3.23e+00  1.54e+00 -8.17e-01 -1.31e+00  1.08e-01 -8.60e-01 -7.19e-02
   9.07e-01 -1.72e+00  7.98e-01 -6.76e-03  1.96e+00 -6.45e-01  3.02e+00
  -5.40e-01  3.32e-02 -7.75e-01  1.06e-01 -4.31e-01  2.30e-01 -7.06e-02
   1.29e+01]]
X_test samples :  [[-3.23e-01  1.06e+00 -4.83e-02 -6.07e-01  1.26e+00 -9.18e-02  1.16e+00
  -1.24e-01 -1.75e-01 -1.64e+00 -1.12e+00  2.03e-01  1.15e+00 -1.80e+00
  -2.47e-01 -6.09e-02  8.47e-01  3.79e-01  8.47e-01  1.86e-01 -2.07e-01
  -4.34e-01 -2.62e-01 -4.67e-02  2.12e-01  8.30e-03  1.08e-01  1.61e-01
   4.00e+01]]
y_train samples :  [0 0 0 0 0 0 0 0 0 0]
y_test samples :  [0 0 0 0 0 0 0 0 0 0]

In [38]:
```

Step 6: Perform feature scaling

From the variables data, it can be observed that the range of values in the 'Amount' column are varying enormously when compared to the rest of the variables. In order to reduce this range, we need to perform standardisation operation using 'StandardScaler' method in python.

Code:

```
sc = StandardScaler()

amount = df['Amount'].values

df['Amount'] = sc.fit_transform(amount.reshape(-1, 1))

print(df['Amount'].head(10))
```

```
0    0.244964
1   -0.342475
2    1.160686
3    0.140534
4   -0.073403
5   -0.338556
6   -0.333279
7   -0.190107
8    0.019392
9   -0.338516
Name: Amount, dtype: float64
```

Step 7: Develop the classification models

In this step, five different types of classification models will be built namely Decision Tree, K-Nearest Neighbours (KNN), Logistic Regression, Support Vector Machine (SVM), Random Forest.

A. Decision Tree

In this algorithm, we have mentioned the 'max_depth' to be '4' which means we are allowing the tree to split four times and the 'criterion' to be 'entropy' which is most similar to the 'max_depth' but determines when to stop splitting the tree. Finally, we have fitted and stored the predicted values into the 'tree_yhat' variable.

Code:

```
tree_model = DecisionTreeClassifier(max_depth = 4, criterion = 'entropy')
tree_model.fit(X_train, y_train)
tree_yhat = tree_model.predict(X_test)
```

Output in Variable Explorer:

Desired variables are created as shown below.

tree_model	tree._classes.DecisionTreeClassifier	1	DecisionTreeClassifier object of sklearn.tree._classes module
tree_yhat	Array of int64	(56962,)	[0 0 0 ... 0 0 0]

B. K-Nearest Neighbour

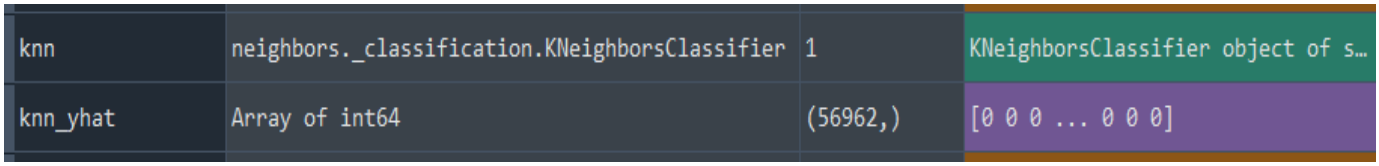
In this algorithm, 'n_neighbors' to be '5'. The value of the 'n_neighbors' is randomly selected but can be chosen optimistically through iterating a range of values, followed by fitting and storing the predicted values into the 'knn_yhat' variable.

Code:

```
n = 5
knn = KNeighborsClassifier(n_neighbors = n)
knn.fit(X_train, y_train)
knn_yhat = knn.predict(X_test)
```

Output in Variable Explorer:

Desired variables are created as shown below.



knn	neighbors._classification.KNeighborsClassifier	1	KNeighborsClassifier object of s…
knn_yhat	Array of int64	(56962,)	[0 0 0 ... 0 0 0]

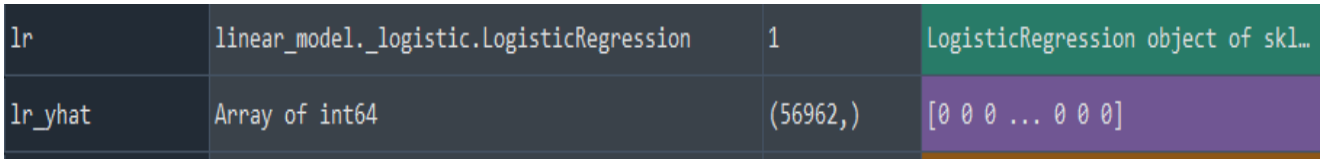
C. Logistic Regression

Code:

```
lr = LogisticRegression()

lr.fit(X_train, y_train)

lr_yhat = lr.predict(X_test)
```

Output in Variable Explorer:

Desired variables are created as shown below.



lr	linear_model._logistic.LogisticRegression	1	LogisticRegression object of skl…
lr_yhat	Array of int64	(56962,)	[0 0 0 ... 0 0 0]

D. Support Vector Machine model (SVM)

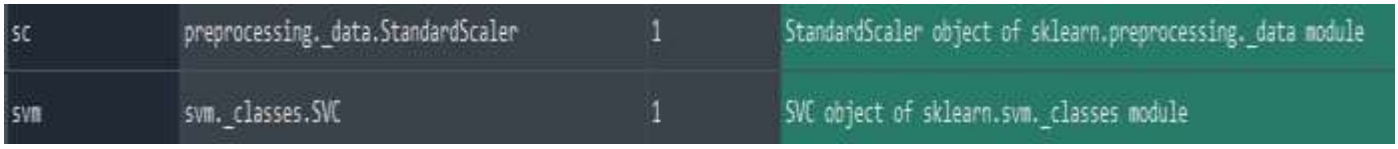
We built the Support Vector Machine model using the 'SVC' algorithm and we didn't mention anything inside the algorithm as we managed to use the default kernel which is the 'rbf' kernel. After that, we stored the predicted values into the 'svm_yhat' after fitting the model.

Code:

```
svm = SVC()

svm.fit(X_train, y_train)

svm_yhat = svm.predict(X_test)
```

Output in Variable Explorer:

Desired variables are created as shown below.



sc	preprocessing._data.StandardScaler	1	StandardScaler object of sklearn.preprocessing._data module
svm	svm._classes.SVC	1	SVC object of sklearn.svm._classes module

E. Random Forest Tree

The next model is the Random forest model. In this, we use 'RandomForestClassifier' algorithm. Also, 'max_depth' to be 4 which means tree will split four times. Finally, fitting and storing the values into the 'rf_yhat'.

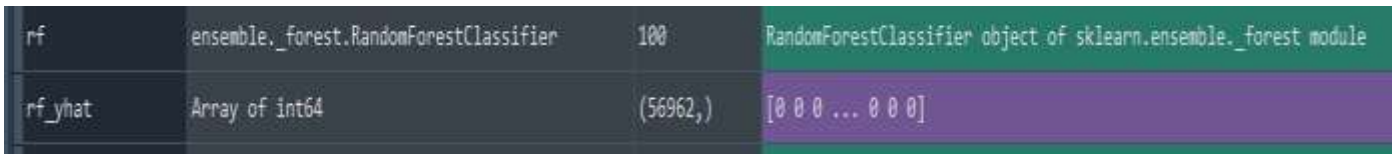
Remember that the main difference between the decision tree and the random forest is that, decision tree uses the entire dataset to construct a single model whereas, the random forest uses randomly selected features to construct multiple models. That's the reason why the random forest model is used versus a decision tree.

Code:

```
rf = RandomForestClassifier(max_depth = 4)

rf.fit(X_train, y_train)

rf_yhat = rf.predict(X_test)
```

Output in Variable Explorer:

Desired variables are created as shown below.



Based on above codes, five different types of classification models are built for interpreting the code in an easy manner. Next step is to evaluate each code using following three techniques, to find the best suited model.

1. Accuracy Score
2. F1 Score
3. Confusion Matrix

Step 8: Evaluate the developed classification models

The evaluation techniques which are being used are provided by scikit-learn package. We have already imported the packages in step 1. Our main objective is to find the best model out of 5 models which have been built in step 7.

1. Accuracy score

Accuracy score is one of the most basic evaluation metrics which is widely used to evaluate classification models.

The accuracy score is calculated simply by dividing the number of correct predictions made by the model by the total number of predictions made by the model (can be multiplied by 100 to transform the result into a percentage). It can generally be expressed as:

Accuracy score = No. of correct predictions / Total no. of predictions

Code:

```
print('ACCURACY SCORE')
print('Accuracy score of the Decision Tree model is {}'.format(accuracy_score(y_test, tree_yhat)))
print('Accuracy score of the KNN model is {}'.format(accuracy_score(y_test, knn_yhat)))
print('Accuracy score of the Logistic Regression model is {}'.format(accuracy_score(y_test, lr_yhat)))
print('Accuracy score of the SVM model is {}'.format(accuracy_score(y_test, svm_yhat)))
print('Accuracy score of the Random Forest Tree model is {}'.format(accuracy_score(y_test, rf_yhat)))
```

```
ACCURACY SCORE
Accuracy score of the Decision Tree model is 0.9993679997191109
Accuracy score of the KNN model is 0.999403110845827
Accuracy score of the Logistic Regression model is 0.9991748885221726
Accuracy score of the SVM model is 0.998735999438222
Accuracy score of the Random Forest Tree model is 0.9992977774656788
```

Based on accuracy score evaluation, all the models have shown 99% accuracy. However, on detailed analysis, KNN model reveals to be the most accurate model and Logistic regression is the least accurate one.

2. F1 Score

The F1 score or F-score is one of the most popular evaluation metrics used for evaluating classification models. It can be simply defined as the harmonic mean of the model's precision and recall. It is calculated by dividing the product of the model's precision and recall by the value obtained on adding the model's precision and recall and finally multiplying the result with 2. It can be expressed as:

F1 score = 2((precision * recall) / (precision + recall))

Code:

```
print('F1 SCORE')
print('F1 score of the Decision Tree model is {}'.format(f1_score(y_test, tree_yhat)))
print('F1 score of the KNN model is {}'.format(f1_score(y_test, knn_yhat)))
print('F1 score of the Logistic Regression model is {}'.format(f1_score(y_test, lr_yhat)))
print('F1 score of the SVM model is {}'.format(f1_score(y_test, svm_yhat)))
print('F1 score of the Random Forest Tree model is {}'.format(f1_score(y_test, rf_yhat)))
```

```
F1 score of the Decision Tree model is 0.8105263157894738
F1 score of the KNN model is 0.8089887640449437
F1 score of the Logistic Regression model is 0.7251461988304092
F1 score of the SVM model is 0.5
F1 score of the Random Forest Tree model is 0.7727272727272727
```

Result of the F1 score evaluation matrix is in line with the Accuracy model i.e. KNN model is the best and logistic regression is the least accurate model.

3. Confusion Matrix

A confusion matrix is a visualization of a classification model that shows how well the model has predicted the outcomes when compared to the original ones. Usually, the predicted outcomes are stored in a variable that is then converted into a correlation table. Using the correlation table, the confusion matrix is plotted in the form of a heat map.

	PREDICTED VALUES	
ACTUAL VALUES	TRUE NEGATIVE	FALSE POSITIVE
	FALSE NEGATIVE	TRUE POSITIVE

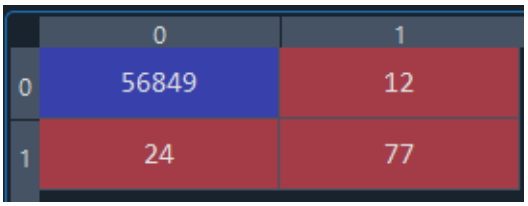
Code:

COMPUTE CONFUSION MATRIX FOR THE MODELS

```
tree_matrix = confusion_matrix(y_test, tree_yhat)      # Decision Tree
knn_matrix = confusion_matrix(y_test, knn_yhat)        # K-Nearest Neighbors
lr_matrix = confusion_matrix(y_test, lr_yhat)          # Logistic Regression
svm_matrix = confusion_matrix(y_test, svm_yhat)        # Support Vector Machine
rf_matrix = confusion_matrix(y_test, rf_yhat)          # Random Forest Tree
```

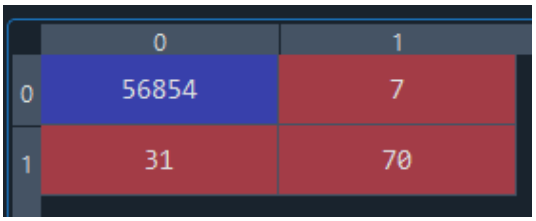
1. Decision tree

Corrected Value Prediction: 56926
Wrong value Prediction: 36
% Accuracy: 99.936%



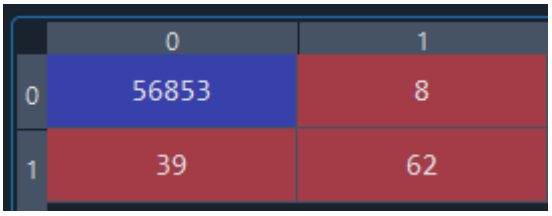
2. K-Nearest Neighbors

Corrected Value Prediction: 56924
Wrong value Prediction: 38
% Accuracy: 99.933%



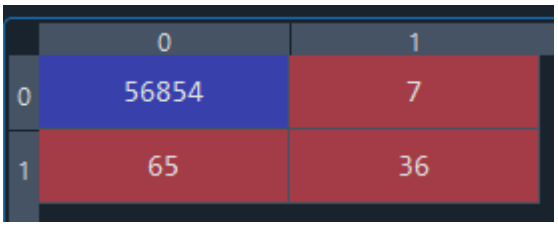
3. Logistic regression

Corrected Value Prediction: 56915
Wrong value Prediction: 47
% Accuracy: 99.917%



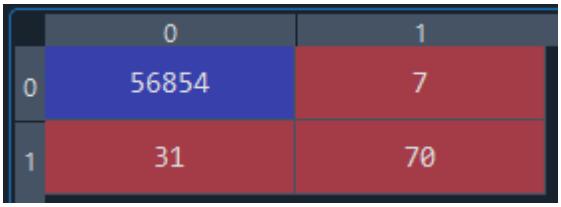
4. Support Vector Machine

Corrected Value Prediction: 56890
Wrong value Prediction: 72
% Accuracy: 99.8735%



5. Random forest tree

Corrected Value Prediction: 56924
Wrong value Prediction: 38
% Accuracy: 99.933%



CONCLUSION

From the above mentioned analysis, following points can be concluded:

- For analysis of Credit Card Fraud detection, 5 Machine Learning models have been built namely,
  - Decision Tree
  - K-NN model
  - Logistic Regression
  - SVM method
  - Random Forest Tree
- For model evaluation, following 3 methods have been employed,
  - Accuracy Score
  - F1 Score
  - Confusion Matrix
- From the above mentioned analysis results, it can be concluded that K-NN model developed based on the input variables has the **BEST ACCURACY**.